Each scripting language has specific words or keywords with specific definitions and usage guidelines. Python is no exception. The fundamental building blocks of any Python program are Python keywords.

In this tutorial, you will get a basic overview of all Python keywords and a detailed discussion of some important keywords that are often used.

Introduction to Python Keywords

Python keywords are unique reserved words with specific meanings and functions that we can apply only to those functions. You will never need to import keywords into your program because they are always there.

Python's built-in methods and classes are not the same as keywords. Built-in methods and classes are still present; however, their use is not limited to keywords

Assigning a specific meaning to Python keywords means that we cannot use them for other purposes in our code. You will get a SyntaxError if you try to do the same. If you try to assign something to a built-in method or type, you won't get a SyntaxError; However, it's still not the smartest idea.

The latest version of Python, namely Python 3.8, contains thirty-five keywords. Here we have shown a complete list of Python keywords for the reader's reference.

- Wrong, expect another import pass
- Not a single pause except a restart
- Real class is finally back
- and continue testing Lambda
- as the definition of nonlocal while
- the state of the world doesn't use it
- elif asynchronous if or exit

The above keywords may change in different versions of Python. Some additions may be added and others may be removed. By writing the following statement in the coding window, you can get a set of keywords in the version you are working on at any time.

Coded

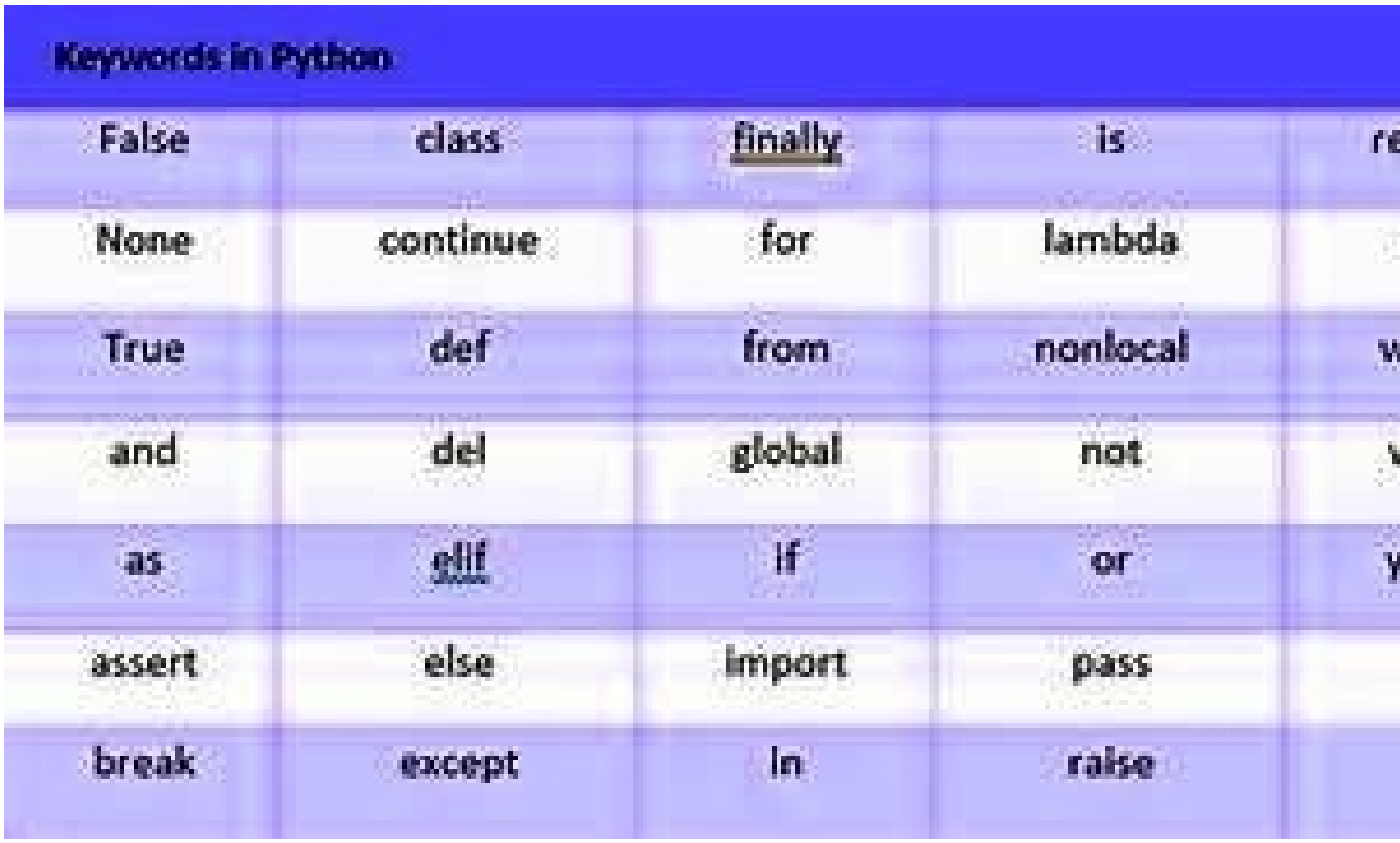
```
# Python program to demonstrate is keyword()
# Import keyword library with lists
import keyword
```

```
# display the full list using "kwlist()".
print("Keyword set in this version:")
print (keyword. twist)
```

Exit:

```
Keyword defined in this version:
['False', 'None', 'True', 'and', 'as', 'assert', 'asynchronous', 'wait', 'abort',
'class', 'continue', 'def', ' del', 'elif', 'else', 'except', 'finally', 'for', 'from',
'global', 'if', 'import', 'in', 'is', 'lambda', "nonlocal", "not", "or", "pass",
"retry", "return", "try", "during", "with", "exit"]
```

By calling help() you can get a list of currently suggested keywords:



Coded:

```
help("keywords")
```

How to identify Python keywords

The Python keyword collection has evolved as new versions have been released. The wait and async keywords, for example, were only introduced with Python 3.7. Additionally, in Python 2.7, print and exec are keywords; However, in Python 3+ they have been converted to built-in methods and are no longer part of the keyword set. In the following

paragraphs, we will introduce several methods to determine whether or not a particular word is a keyword in Python.

Write code in an IDE with syntax highlighting

There are many excellent Python IDEs available. They all highlight keywords to distinguish them from other terms in the code. This feature will help you immediately identify Python keywords while coding so that you don't use them incorrectly.

Checking keywords using a script in the REPL

There are several ways to discover acceptable Python keywords, as well as more information about them in Python REPL.

Find syntax error

Finally, if you get a SyntaxError when you try to allocate memory, call a method with it, or do anything else with it, and it's not allowed, it's probably a word -key. It's a little harder to see, but it still allows Python to tell you if you're using a keyword incorrectly.

Python Keywords and Their Uses:

The following sections categorize Python keywords into titles based on how often they are used. For example, the first category includes all keywords used as values, while the next group includes keywords used as operators. These classifications will help you understand how keywords are used and help you organize Python's vast collection of keywords.

Some of the terms mentioned in the following segment may not be familiar to you. They are explained here and you should understand what they mean before continuing:
The Boolean estimate of a variable is called veracity. Value truth indicates whether the value of a variable is true or false.

In the Boolean paradigm, truth refers to any variable whose value is true. Pass the element as input to bool() to check if it is true. If True is returned, the item's value is true. Examples of real values are strings and lists that are not empty, non-zero numbers, and many other objects.

False refers to any element of a Boolean expression that evaluates to false. Pass an element as input to bool() to check if it is false. If False is returned, the element value is false. Examples of false values are " ", 0, { }, and [ ].

Keyword values: True, False, None.

This example uses three Python keywords as values. These are singular values that can be reused endlessly and which correspond to the same entity each time. These meanings are likely to be encountered and used frequently.

Keywords true and false:

These keywords are typed in lowercase letters in common computer languages (true and false); however, in Python, they are capitalized every time. In a Python script, the True Python keyword represents a true logical state. False is a keyword equivalent to True, except that it has a negative Boolean status of false.

True and False are keywords that can be assigned to variables or parameters and compared directly.

Coded

```
print(4==4)
print(6>9)
print (True or False)
print(99)
print (True and False)
```

Exit:

```
TRUE
LIE
TRUE
TRUE
LIE
LIE
```

Since the first, third, and fourth statements are true, the interpreter assigns the value “True” to these statements and “False” to the other statements. True and False in Python are equal to 1 and 0. We can use an illustration to support this statement:

Coded

```
print(True==3)
print(False==0)
print (True + True + True)
```

Exit:

LIE
TRUE
3

Keyword “No”

None is a Python keyword meaning "nothing". None are referred to in various computer languages as null, null, or undefined.

If a function does not have a return clause, it will produce None as its default output:

Coded

```
print (No == 0)
print(No == """)
print (No == False)
A = no
B=n
print(A==B)
```

Exit:

LIE
LIE
LIE

TRUE

If no_return_function returns nothing, it will simply return None. None are provided by functions that do not match the expression returned in the program flow. Consider the following scenario:

Coded

```
protection no_return_function():
number1 = 10
number2 = 20
addition = number1 + number2

number = no_return_function()
print (number)
```

None

This program has a with_return function that performs several operations

and contains a return expression. Therefore, if we print a number, we get None, which is the default when there is no return statement. Here is an example showing this:

Coded

```
def with_return(num):
if num% 4 == 0:
return False

number = with_return(67)
print (number)
```

Exit:

None

Keyword operators: and or not, in, is

Several Python keywords are used as operators to perform mathematical operations. In many other computer languages, these operators are represented by symbols such as &, | AND! These are all keyword operations in Python:

Mathematical operations Operations in other languages Python keyword
E, ⊥ && e
OR, ⊦ || OR
NO, ¬! NO
CONTAINS, ∈ inches
IDENTITY === yes

The authors created Python programming with clarity in mind. As a result, many operators in other computer languages that use symbols in Python are English words called keywords.

- ### Keyword and:

The Python keyword determines whether the left and right operands are true or false. The result will be true if both components are true. If one of them is false, the result will also be false:

- The truth table for e
- X Y X and Y
- True True True
- False True False
- True False False
- Lie, lie, lie

AND

It should be noted that operator results are not always true or false. This is due to the particular behavior of e. Instead of processing the input to produce the corresponding Boolean values, it simply returns whether it is false or true. The output of the a and expression can be used with an if conditional clause or passed to bool() to produce an obvious True or False answer.

- ## Keyword or

The or keyword in Python is used to check if at least one of the inputs is true. If the first argument is true, the operation returns it; otherwise, the second argument is returned:

OR

Like the and keyword, the or keyword does not modify its entries to corresponding Boolean values. Instead, results are determined by whether they are true or false.

- The truth table for o
- X Y X or Y
- True True True
- True False True
- False True True
- Lie, lie, lie
- ## Keyword “no”

The not keyword in Python is used to get the opposite Boolean value of a variable:

The not keyword is used to modify the logical interpretation or produce conditions or other logical equations. No, unlike, and, and or, specifies a specific logical state, True or False, and therefore returns the opposite value.

The truth table for no
X is not
Right wrong
False truth

Coded

Lies and truth
Lies or truth
not true

Exit:

LIE
TRUE
LIE

Keyword

The keyword in Python is a robust constraint check, also known as the membership operator. If you give it an element to search for and a container or series to search for, it will return True or False, depending on whether the specified element was in that container:

V:

Testing for a specific character in a string is a good example of using the keyword in:

Coded:

```
container = "Javapoint"
print("p" in container)
print("P" in container)
```

Exit:

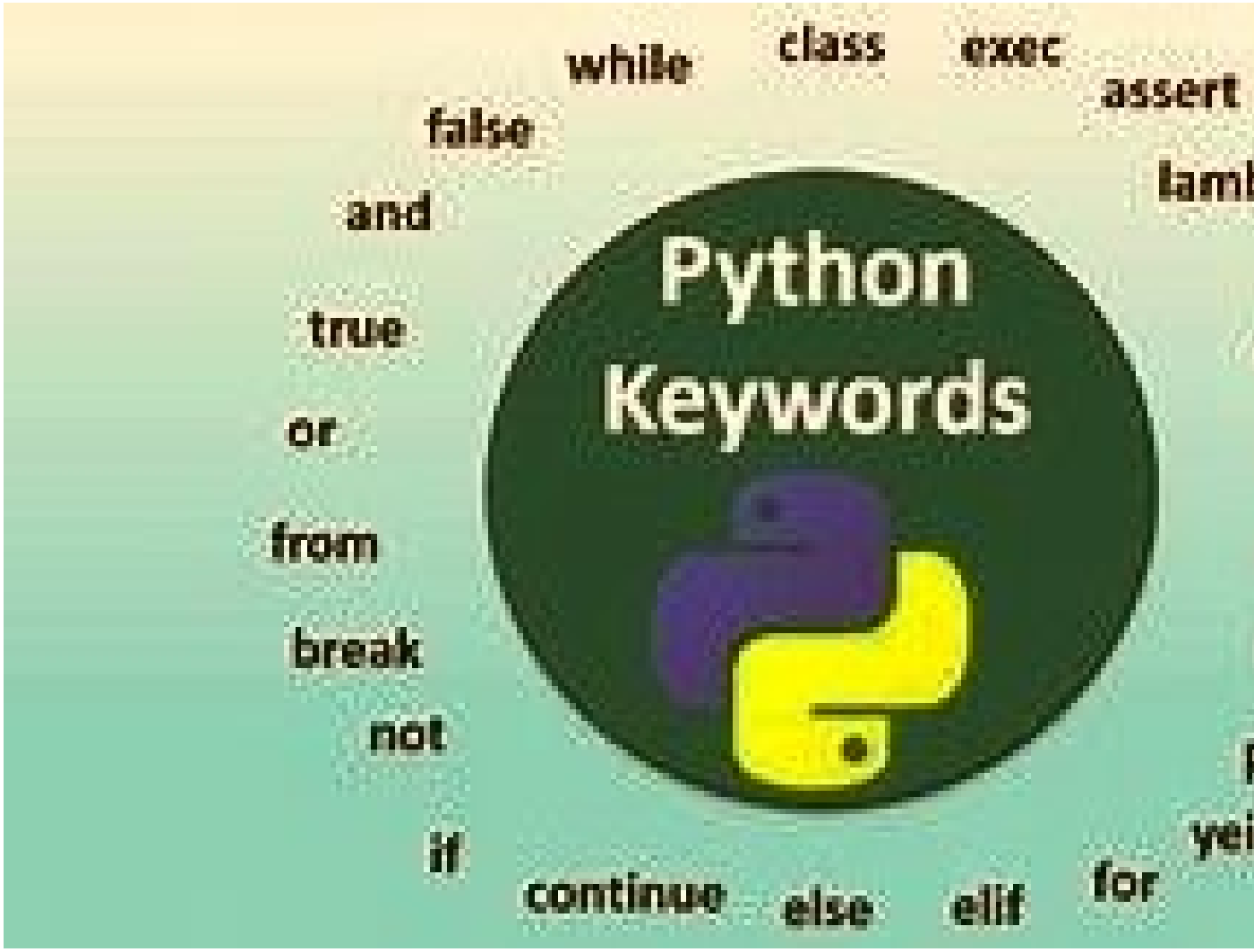
TRUE
LIE

Lists, dictionaries, tuples, strings, or any data type that uses the __contains__() method, or we can iterate over them, will work with the in keyword.

Keyword

In Python, it is used to verify the identity of objects. The == operator is used to determine the identity of two arguments. It also determines whether the two arguments refer to a single object.

When the objects are the same, returns True; otherwise, False is returned.



Coded

```
print(The truth is the truth)
print (False equals True)
print (No - not No)
print((9 + 5) = (7 * 2))
```

Exit:

```
TRUE
LIE
LIE
TRUE
```

True, False, and None are the same thing in Python because there is only one version.

Coded:

```
print([]==[])
print([] is [])
print({}=={})
print({} is {})
```

Exit:

```
TRUE
LIE
TRUE
LIE
```

An empty dictionary or list is similar to another empty dictionary. However, they are not identical objects because they are stored in memory independently. Indeed, the list and the dictionary are modifiable.

Coded:

```
print(''=='')
print(''is''')
```

Exit:

```
TRUE
TRUE
```

Strings and tuples, unlike lists and dictionaries, are immutable. Therefore, even two identical strings or tuples are identical. Both belong to a single memory region.

Non-local Keyword:

Using non-local keywords is very similar to using global keywords. The nonlocal keyword is intended to indicate that a variable in a function is inside the function, that is, a nested function is simply not local to it, implying that it is in the external function. We need to define a nonlocal parameter using Nonlocal if we ever need to change its value in a nested function. Otherwise, the nested function creates a local variable using this header. The following example will help us clarify this.

Coded:

```
protect the_outer_function():
variable = 10
protect the_inner_function():
non-local variable
variable = 14
```

```
print("Value inside internal function:", var)
the_internal_function()
print("Value inside external function:", var)

the_external_function()
```

Exit:

```
The value inside the internal function: 14
The value inside the external function: 14
```

In this case, the_inner_function() is placed inside the_outer_function.

The outer_function has a variable called var. As you may have noticed, Var is not a global variable. Therefore, if we want to modify it in the_inner_function(), we need to declare it using non-local.

As a result, the variable was updated in the nested function the_inner_function, as evidenced by the results. Here's what happens if you don't use the nonlocal keyword:

Coded:

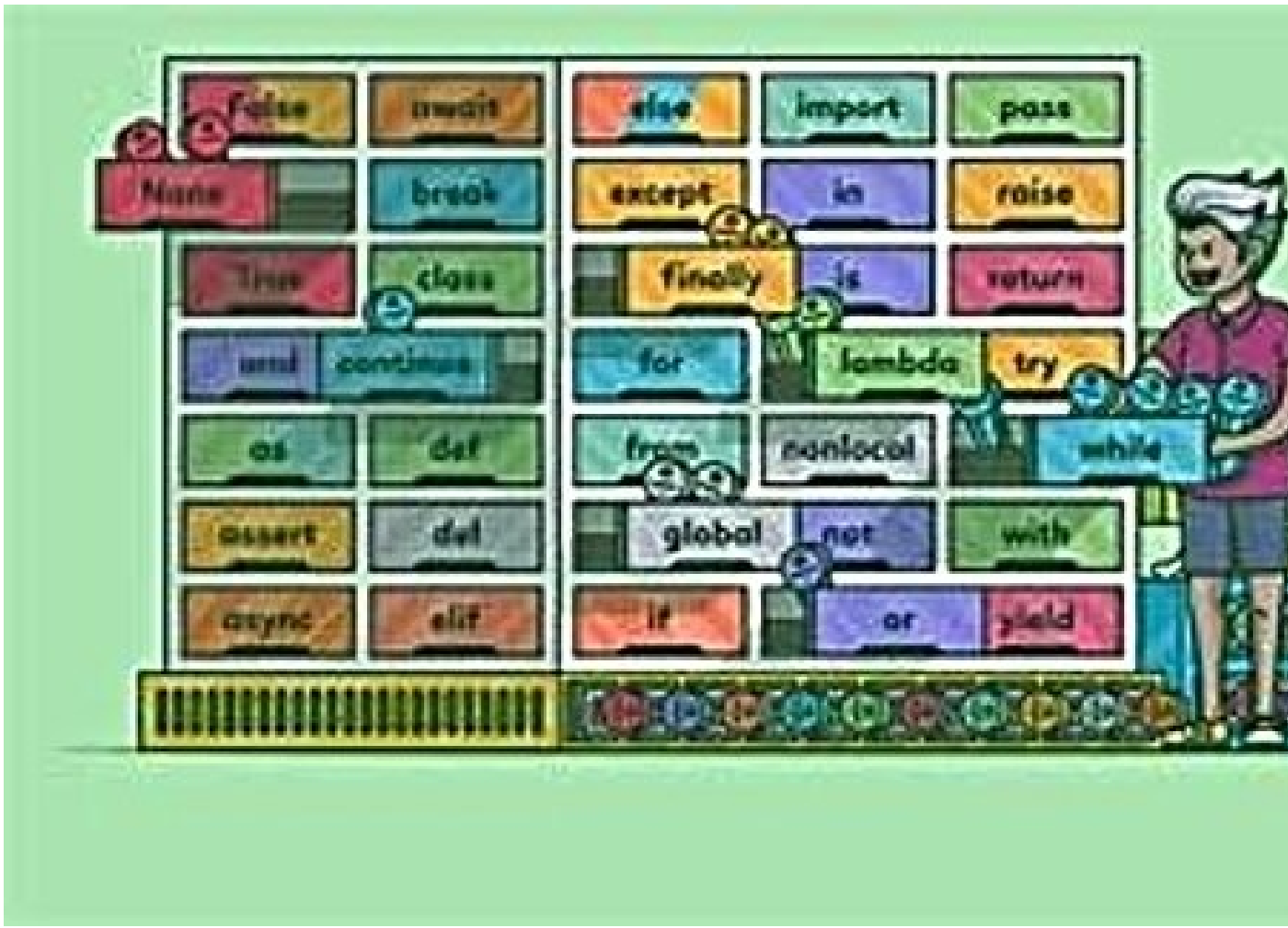
```
protect the_outer_function():
variable = 10
protect the_inner_function():
variable = 14
print("Value inside internal function:", var)
the_internal_function()
print("Value inside external function:", var)

the_external_function()
```

Exit:

```
The value inside the internal function: 14
The value inside the external function: 10
```

Iteration Keywords: for, while, Break, continue.

The iterative process and loops are important fundamentals of programming. There are several keywords for creating and using loops in Python. They will be used and seen in almost every Python program. Knowing how to use them correctly can help you become a better Python developer.



Keyword:

The for loop is by far the most popular loop in Python. It is created by combining two Python keywords. They are for and for, as explained above.

- **While Keyword:**

The while loop in Python uses the term while and works similarly to while loops in other computer languages. The block after the while clause will be repeated repeatedly until the condition following the while keyword is false.

- **Keyword Skipping:**

If you want to quickly break out of a loop, use the Break keyword. We can use this keyword in for and while loops.

- **Keyword “Continue”**

You can use Python's continue keyword if you want to move to the next iteration of the loop. The continue keyword, as in many other computer languages, allows you to stop the execution of the current iteration of a loop and move on to the next one.

Coded:

```
# Program to show keyword usage for, while, break, continue
for me in the range (15):

print(i + 4, end = """)

# break the loop when i = 9
if I == 9:
to break
Press()

# cycle from 1 to 15
i = 0 # initial condition
while I am < 15 years old:

# When i is 9, the loop will move to the next iteration using continue. It
will not print
if I == 9:
i+= 3
keep on going
Moreover:
# when i is not equal to 9, add 2 and print the value
print(i + 2, end = """)

I += 1
```

Exit:

```
4 5 6 7 8 9 10 11 12 13
2 3 4 5 6 7 8 9 10 14 15 16
```

Exception handling keywords: try, except, raise, finally, and assert

Try: This keyword is intended for exception handling and is used in conjunction with the "exception" keyword to handle problems in a program. If an error occurs, the program inside the "try" block is checked, but the code in that block is not executed.

Except: As noted above, this works in conjunction with exception handling.

finally: Regardless of the result of the "test" section, the "finally" field will be implemented every time.

raise - The raise keyword can be used to specifically raise an exception.

Statement: This method is used for troubleshooting. Often used to verify code correctness. Nothing happens if the expression is interpreted as true; however, if it is false, an "AssertionError" message is generated. You can also print the result with an error followed by a comma.

Coded:

```
# initialization of numbers
variable1 = 4
variable2 = 0

# The exception was thrown in the try section.
Attempt:
d = var1 // var2 # This will cause a divide by zero exception.
print(s)
# This section will handle the exception thrown in the try block
except ZeroDivisionError:
print("Cannot divide by zero")
Finally:
# Whether an exception occurred or not, this block will be executed every time
print("It's finally inside the block")
# using the assert keyword we will check if var2 is equal to 0
print("The value of var1/var2 is:")
Assert var2 != 0, "Error dividing by 0"
print (var1 / var2)
```

Exit:

```
You can't divide by zero
It's finally inside the block
var1/var2 value:
--------------------------------------------------- ------------------------
AssertionError Traceback (most recent call is last)
Enter in [44], in ()
        15 # Using the Assert keyword we will check if var2 is equal to 0
        16 print("Value var1/var2:")
---> 17 Assert var2 != 0, “Division by 0 error”
        18 seals (var1/var2)

AssertionError: division by 0 error
```

Keyword “pass”

In Python, a null clause is called a pass. It serves as a substitute for something else. When it starts, nothing happens.

Let's say we have a feature that hasn't been coded yet, but we want to build it long-term. If we write it in the middle of the code,

Coded:

def pass_function(arguments):

Exit:

pass_protection function (arguments):

IndentationError:

Indented block expected after function definition on line 1.

as shown, an IndentationError will be raised. Instead, we use the pass command to create an empty container.

Coded:

def pass_function(arguments):
to go up

We can also use the pass keyword to create an empty class.

Coded:

class passed_class:
to go up
returns the keyword

The return expression is used to exit the function and generate a result.

The None keyword is returned by default unless a value is specifically returned. The following example demonstrates this.

Coded:

protection func_with_return():
variable = 13
return var

def func_with_no_return():
variable = 10

```
print(function_with_return())
print(function_with_no_return())
```

Exit:

```
13
None
```

Keyword of

The del keyword is used to remove any reference to an object. In Python, each entity is an object. We can use the del command to remove a reference to a variable.

Coded:

```
var1 = var2 = 5
of var1
print(var2)
print (var1)
```

Exit:

```
5
--------------------------------------------------- -------------------------
NameError Traceback (most recent call is last)
Enter in [42], in ()
        2 of variable 1
        3 seals (var2)
----> 4 prints (var1)
```

NameError: Name "var1" is not defined

We see that the reference to the variable var1 has been deleted. Result: it is no longer recognized. However, var2 still exists.

You can also delete entries from a collection such as a list or a dictionary using del:

Coded

```
list_ = ['A', 'B', 'C']
delete list_[2]
print(list_)
```

Exit:

```
['A B']
```